Remarks

I. Background

Responsive to a Final Rejection mailed January 12, 2006, applicants on February 9, 2006 filed a Request for Reconsideration that demonstrated with various arguments the deficiency of the Final Rejection, one of those arguments showing that the primary reference, U.S. Patent No. 6,345,302 to Bennett et al. ("Bennett") was nonenabled. An Advisory Action was mailed February 28, 2006, stating that the Request for Reconsideration had been considered but does not place the application in condition for allowance because:

> The claims, as discussed in the Final Office Action, are met by the prior art. Further discussion will be provided in due course. However, the examiner would like to point out a fact that may not have been considered when Applicant allged (*sic*) that Bennett did not provide an enabled teaching. That is, TCP is associated with a re-transmittion (*sic*) mechanism causing any lost packet to be re-transmitted, and therefore any ACK packet sent out would trusfully (*sic*) reflect completion of all the prior sent packets.

II. Applicants' Response

Applicants' arguments demonstrating the deficiency of the Final Rejection will not be reiterated here. The Examiner's statement regarding TCP retransmission, however, merits comment.

Applicants are well aware of the retransmission mechanism for TCP and note that TCP retransmission depends upon the failure of the sender of data to receive an ACK within a certain time period, and that Bennett thwarts this retransmission mechanism by automatically sending ACKs despite not having received all the data.

As noted in Stevens, "TCP/IP Illustrated, Volume 1, The Protocols," which was cited in Bennett and the present application:

> TCP provides a reliable transport layer. One of the ways it provides reliability is for each end to acknowledge the data it receives from the other end. But data segments and acknowledgements can get lost. *TCP handles this by setting a timeout when it sends data, and if the data isn't acknowledged when the timeout expires, it retransmits the data.*[1]

[1] Richard Stevens, "TCP/IP Illustrated, Volume 1, The Protocols" (1994), page 297, lines 2-6, emphasis added. For the Examiner's convenience, a copy of page 297 of Stevens is enclosed.

As noted previously, Bennett claims to automatically send an ACK for a datagram upon verifying the checksum for the datagram.[2] But the TCP protocol specifies that sending an ACK signals to the receiver of the ACK that all the data prior to that ACK number has been successfully received by the sender of the ACK. Because Bennett sends ACKs despite not having received all the data signified by its ACKs, the timeout does not expire, and the lost data is not retransmitted. As stated by Stevens:

> *...the acknowledgement number in the TCP header means that the sender has successfully received up through but not including that byte.* There is currently no way to acknowledge selected pieces of the data stream. For example, if bytes 1-1024 are received OK, and the next segment contains bytes 2049-3072, the receiver cannot acknowledge this new segment. All it can send is an ACK with 1025 as the acknowledgement number."[3]

According to Bennett's preferred embodiment, however, the "NIC 2000" would automatically send an ACK with 3073 as the acknowledgement number for the example of Stevens, assuming that the checksum for bytes 2049-3072 was valid. This would indicate to the receiver of that ACK that all data up through byte 3072 was successfully received by the sender of the ACK, even though bytes 1025-2048 were never in fact received.

What applicants neglected to mention in the prior Request for Reconsideration is that the TCP retransmission mechanism would fail to retransmit lost packets because the retransmission mechanism would be fooled by Bennett's automatically generated ACK into believing that all the bytes prior to the ACK number had been successfully received. As stated in Comer, "Internetworking with TCP/IP," Volume 1, (1991), which was repeatedly referenced in Bennett:

> We have said that the reliable stream delivery service guarantees to deliver a stream of data sent from one machine to another without duplication or data loss. The question arises: "How can protocol software provide reliable transfer if the underlying communication system offers only unreliable packet delivery?" The answer is complicated, but most reliable protocols use a single fundamental technique known as *positive*

---

[2] See, e.g., column 12, lines 7-11; column 16, lines 19-26.

[3] Stevens, page 226, lines 34-38, emphasis added. For the Examiner's convenience, a copy of page 226 of Stevens was enclosed with the prior Request for Reconsideration.

> *acknowledgement with retransmission.* The technique requires the recipient to communicate with the source, sending back an *acknowledgement* message as it receives data. The sender keeps a record of each packet it sends and waits for an acknowledgement before sending the next packet. The sender also starts a timer when it sends a packet and *retransmits* a packet if the timer expires before an acknowledgement arrives.[4]

Such a timeout and retransmission is depicted in Figure 12.2 of Comer, for which the caption reads: "Timeout and retransmission that occurs when a packet is lost."[5] Moreover, as noted on page 187 of Comer:

> *Acknowledgements always specify the sequence number of the next octet that the receiver expects to receive.*
>
> The TCP acknowledgement scheme is called *cumulative* because it reports how much of the stream has accumulated.[6]

Stated differently, sending an ACK upon receiving a packet even though a prior packet may not have been received violates the TCP protocol. Moreover, because Bennett teaches automatically generating ACKs upon verification of a checksum, the ACKs signaling to the sender that all prior data in the stream has been received without regard to whether this is true, Bennett's ACKs would circumvent the timeout and retransmission mechanism that TCP relies upon, causing data corruption.

In addition to the glaring failures of Bennett noted above and in the prior Request for Reconsideration, one of ordinary skill in the art would have recognized other failures of Bennett's ACK generation mechanism. For example, because Bennett's card 2000 automatically generates ACKs, it neglects TCP's requirement of assured delivery of data. This is because the card 2000 at most performs partial protocol processing despite sending ACKs, and even for checksummed packets that arrived in order there is still the opportunity for Bennett's CPU 10 to discard a packet due to other reasons, so that the ACK sent by the card 2000 is erroneous. For example, header errors not caught by the checksum would presumably be recognized by CPU 10, causing the associated data to be

---

[4] Douglas E. Comer, "Internetworking with TCP/IP," Volume 1, (1991), page 173, line 34 – page 174, line 2, underline added, italics in original. For the Examiner's convenience, a copy of pages 174 - 175 and 187 of Comer is enclosed.
[5] Comer, Chapter 12, "Reliable Stream Transport Service (TCP)," Figure 12.2, page 175.
[6] Comer, page 187, lines 16-40, emphasis in original.

discarded. Conversely, should CPU 10 or TCP Process 91 crash after an ACK had been automatically generated by card 2000 but before that data could be processed by TCP Process 91, that data would be lost and not retransmitted. Similarly, the TCP Process 91 running on CPU 10 may also discard the data due to resource limitations, with TCP Process 91 assuming that no ACK for the data has been sent and that the data would be retransmitted once a timeout occurs at the sender.

The primary purpose of the TCP protocol is the guaranteed delivery of data. Bennett's foremost objective is also to "provide an improved method and apparatus for efficiently operating a reliable communication protocol in a computer network."[7] Yet the invention actually disclosed by Bennett would destroy the reliability and guaranteed delivery of data, thwarting the primary purposes of TCP and Bennett. For at least this reason, Bennett does not anticipate or render obvious any of the claims of the present application. Indeed, Bennett instead demonstrates a long-standing need for the invention defined by the present claims, and a failure of others in their approach to solving that need.

Applicants respectfully request the Examiner to reconsider the Final Rejection, because the Final Rejection has not presented a *prima facie* case of anticipation or obviousness for any of the claims, and because the primary reference relied upon for that rejection is nonenabled. As such, applicants respectfully assert that the Final Rejection should be withdrawn, and a Notice of Allowance provided.

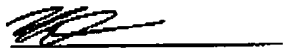
Respectfully submitted,

Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

CERTIFICATE OF FACSIMILE TRANSMISSION

I hereby certify that this correspondence is being transmitted via facsimile to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313, telephone number (571) 273-8300, on March 15, 2006.

Date: 3-15-06



Mark Lauer

[7] Summary of the Invention, column 1, line 65 – column 2, line 1.

UNIX is a technology trademark of X/Open Company, Ltd.

The publisher offers discounts on this book when ordered in quantity for special sales.
For more information please contact:

Corporate & Professional Publishing Group
Addison-Wesley Publishing Company
One Jacob Way
Reading, Massachusetts 01867

Library of Congress Cataloging-in-Publication Data
Stevens, W. Richard
TCP/IP Illustrated: the protocols/W. Richard Stevens.
p. cm. — (Addison-Wesley professional computing series)
Includes bibliographical references and index.
ISBN 0-201-63346-9 (v. 1)
1.TCP/IP (Computer network protocol) I. Title. II. Series.
TK5105.55S74 1994
004.6'2—dc20

Text printed on recycled and acid-free paper.
ISBN 0-201-63346-9
13 1415161718 MA 02 01 00 99
13th Printing January 1999

21

TCP Timeout and Retransmission

21.1 Introduction

TCP provides a reliable transport layer. One of the ways it provides reliability is for each end to acknowledge the data it receives from the other end. But data segments and acknowledgments can get lost. TCP handles this by setting a timeout when it sends data, and if the data isn't acknowledged when the timeout expires, it retransmits the data. A critical element of any implementation is the timeout and retransmission strategy. How is the timeout interval determined, and how frequently does a retransmission occur?

We've already seen two examples of timeout and retransmission: (1) In the ICMP port unreachable example in Section 6.5 we saw the TFTP client using UDP employing a simple (and poor) timeout and retransmission strategy: it assumed 5 seconds was an adequate timeout period and retransmitted every 5 seconds. (2) In the ARP example to a nonexistent host (Section 4.5), we saw that when TCP tried to establish the connection it retransmitted its SYN using a longer delay between each retransmission.

TCP manages four different timers for each connection.

1. A *retransmission* timer is used when expecting an acknowledgment from the other end. This chapter looks at this timer in detail, along with related issues such as congestion avoidance.
2. A *persist* timer keeps window size information flowing even if the other end closes its receive window. Chapter 22 describes this timer.
3. A *keepalive* timer detects when the other end on an otherwise idle connection crashes or reboots. Chapter 23 describes this timer.
4. A *2MSL* timer measures the time a connection has been in the TIME_WAIT state. We described this state in Section 18.6.

Library of Congress Cataloging-in-Publication Data

Comer, Douglas E.
Internetworking with TCP/IP / Douglas E. Comer. -- 2nd ed.
p. cm.
Includes bibliographical references (v. 1, p.) and index.
Contents: vol. 1. Principles, protocols, and architecture.
ISBN 0-13-468505-9 (v. 1)
1. Computer networks. 2. Computer network protocols. 3. Data transmission systems. I. Title.
TK5105.5.C69 1991
004.6--dc20 90-7829
CIP

Editorial/production supervision: Joe Scordato
Cover design: Karen Stephens
Cover illustration: Jim Kinstrey
Manufacturing buyers: Linda Behrens and Patrice Fraccio

Printed in the United States of America

10 9 8 7 6 5 4 3

ISBN 0-13-468505-9

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S. A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

receiving application program as soon as they have been received and verified. The protocol software is free to divide the stream into packets independent of the pieces the application program transfers. To make transfer more efficient and to minimize network traffic, implementations usually collect enough data from a stream to fill a reasonably large datagram before transmitting it across an internet. Thus, even if the application program generates the stream one octet at a time, transfer across an internet may be quite efficient. Similarly, if the application program chooses to generate extremely large blocks of data, the protocol software can choose to divide each block into smaller pieces for transmission.

For those applications where data should be delivered even though it does not fill a buffer, the stream service provides a *push* mechanism that applications use to force a transfer. At the sending side, a push forces protocol software to transfer all data that has been generated without waiting to fill a buffer. When it reaches the receiving side, the push causes TCP to make the data available to the application without delay. The reader should note, however, that the push function only guarantees that all data will be transferred; it does not provide record boundaries. Thus, even when delivery is forced, the protocol software may choose to divide the stream in unexpected ways.

• *Unstructured Stream.* It is important to understand that the TCP/IP stream service does not honor structured data streams. For example, there is no way for a payroll application to have the stream service mark boundaries between employee records, or to identify the contents of the stream as being payroll data. Application programs using the stream service must understand stream content and agree on stream format before they initiate a connection.

• *Full Duplex Connection.* Connections provided by the TCP/IP stream service allow concurrent transfer in both directions. Such connections are called *full duplex*. From the point of view of an application process, a full duplex connection consists of two independent streams flowing in opposite directions, with no apparent interaction. The stream service allows an application process to terminate flow in one direction while data continues to flow in the other direction, making the connection *half duplex*. The advantage of a full duplex connection is that the underlying protocol software can send control information for one stream back to the source in datagrams carrying data in the opposite direction. Such *piggybacking* reduces network traffic.

12.4 Providing Reliability

We have said that the reliable stream delivery service guarantees to deliver a stream of data sent from one machine to another without duplication or data loss. The question arises: "How can protocol software provide reliable transfer if the underlying communication system offers only unreliable packet delivery?" The answer is complicated, but most reliable protocols use a single fundamental technique known as *positive acknowledgement with retransmission*. The technique requires a recipient to communicate with the source, sending back an *acknowledgement* message as it receives data. The sender keeps a record of each packet it sends and waits for an acknowledgement

before sending the next packet. The sender also starts a timer when it sends a packet and *retransmits* a packet if the timer expires before an acknowledgement arrives.

Figure 12.1 shows how the simplest positive acknowledgement protocol transfers data.

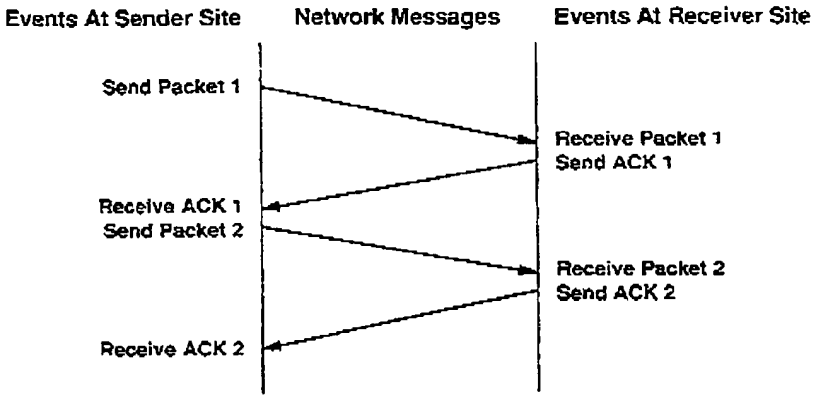


Figure 12.1 A protocol using positive acknowledgement with retransmission in which the sender awaits an acknowledgement for each packet sent. Vertical distance down the figure represents increasing time and diagonal lines across the middle represent network packet transmission.

In the figure, events at the sender and receiver are shown on the left and right. Each diagonal line crossing the middle shows the transfer of one message across the network.

Figure 12.2 uses the same format diagram as Figure 12.1 to show what happens when a packet is lost or corrupted. The sender starts a timer after transmitting a packet. When the timer expires, the sender assumes the packet was lost and retransmits it.

The final reliability problem arises when an underlying packet delivery system duplicates packets. Duplicates can also arise when networks experience high delays that cause premature retransmission. Solving duplication requires careful thought because both packets and acknowledgements can be duplicated. Usually, reliable protocols detect duplicate packets by assigning each packet a sequence number and requiring the receiver to remember which sequence numbers it has received. To avoid confusion caused by delayed or duplicated acknowledgements, positive acknowledgement protocols send sequence numbers back in acknowledgements, so the receiver can correctly associate acknowledgements with packets.

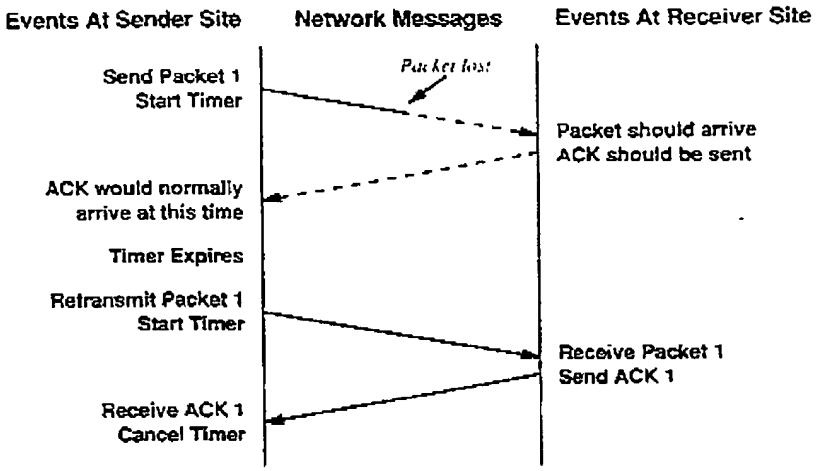


Figure 12.2 Timeout and retransmission that occurs when a packet is lost. The dotted lines show the time that would be taken by the transmission of a packet and its acknowledgement, if the packet was not lost.

12.5 The Idea Behind Sliding Windows

Before examining the TCP stream service, we need to explore an additional concept that underlies stream transmission. The concept, known as a *sliding window*, makes stream transmission efficient. To understand the motivation for sliding windows, recall the sequence of events that Figure 12.1 depicts. To achieve reliability, the sender transmits a packet and then waits for an acknowledgement before transmitting another. As Figure 12.1 shows, data only flows between the machines in one direction at any time, even if the network is capable of simultaneous communication in both directions. The network will be completely idle during times that machines delay responses (e.g., while machines compute routes or checksums). If we imagine a network with high transmission delays, the problem becomes clear:

> *A simple positive acknowledgement protocol wastes a substantial amount of network bandwidth because it must delay sending a new packet until it receives an acknowledgement for the previous packet.*

The sliding window technique is a more complex form of positive acknowledgement and retransmission than the simple method discussed above. Sliding window protocols use network bandwidth better because they allow the sender to transmit multiple

12.15 Acknowledgements And Retransmission

Because TCP sends data in variable length segments, and because retransmitted segments can include more data than the original, acknowledgements cannot easily refer to datagrams or segments. Instead, they refer to a position in the stream using the stream sequence numbers. The receiver collects data octets from arriving segments and reconstructs an exact copy of the stream being sent. Because segments travel in IP datagrams, they can be lost or delivered out of order; the receiver uses the sequence numbers to reorder segments. At any time, the receiver will have reconstructed zero or more octets contiguously from the beginning of the stream, but may have additional pieces of the stream from datagrams that arrived out of order. The receiver always acknowledges the longest contiguous prefix of the stream that has been received correctly. Each acknowledgement specifies a sequence value one greater than the highest octet position in the contiguous prefix it received. Thus, the sender receives continuous feedback from the receiver as it progresses through the stream. We can summarize this important idea:

> *Acknowledgements always specify the sequence number of the next octet that the receiver expects to receive.*

The TCP acknowledgement scheme is called *cumulative* because it reports how much of the stream has accumulated. Cumulative acknowledgements have both advantages and disadvantages. One advantage is that acknowledgements are both easy to generate and unambiguous. Another advantage is that lost acknowledgements do not necessarily force retransmission. A major disadvantage is that the sender does not receive information about all successful transmissions, but only about a single position in the stream that has been received.

To understand why lack of information about all successful transmissions makes the protocol less efficient, think of a window that spans *5000* octets starting at position *101* in the stream, and suppose the sender has transmitted all data in the window by sending five segments. Suppose further that the first segment is lost, but all others arrive intact. The receiver continues to send acknowledgements, but they all specify octet *101*, the next highest contiguous octet it expects to receive. There is no way for the receiver to tell the sender that most of the data for the current window has arrived.

When a timeout occurs at the sender's side, the sender must choose between two potentially inefficient schemes. It may choose to retransmit all five segments instead of the one missing segment. Of course, when the retransmitted segment arrives, the receiver will have correctly received all data from the window, and will acknowledge that it expects octet *5101* next. However, that acknowledgement may not reach the sender quickly enough to prevent the unnecessary retransmission of other segments from the window. If the sender follows accepted implementation policy and retransmits only the first unacknowledged segment, it must wait for the acknowledgement before it can decide what and how much to send. Thus, it reverts to a simple positive acknowledgement protocol and may lose the advantages of having a large window.